pQuant: Towards Effective Low-Bit Language Models via Decoupled Linear Quantization-Aware Training

Anonymous ACL submission

Abstract

Quantization-Aware Training from scratch has emerged as a promising approach for building efficient large language models (LLMs) with extremely low-bit weights (sub 2-bit), which can offer substantial advantages for edge deployment. However, existing methods still fail to achieve satisfactory accuracy and scalability. In this work, we identify a parameter democratization effect as a key bottleneck: the sensitivity of all parameters becomes homogenized, severely limiting expressivity. To address this, we propose pQuant, a method that decouples parameters by splitting linear layers into two specialized branches: a dominant 1-bit branch for efficient computation and a compact high-precision branch dedicated to preserving the most sensitive parameters. Through tailored feature scaling, we explicitly guide the model to allocate sensitive parameters to the high-precision branch. Furthermore, we extend this branch into multiple, sparsely-activated experts, enabling efficient capacity scaling. Extensive experiments indicate our pQuant achieves state-of-the-art performance in extremely low-bit quantization. Our code is available at: `https://anonymous.4open.science/r/anonymous-2026-B94C`

1 Introduction

Large Language Models (LLMs) (Touvron et al., 2023a; Yang et al., 2024; Team et al., 2023; Achiam et al., 2023) have achieved remarkable performance across diverse natural language processing tasks. However, their massive computational and memory demands pose significant challenges for practical deployment, especially on resource-constrained devices (Dettmers et al., 2022). Quantization, which reduces the numerical precision of model weights and activations, has become a key technique to mitigate these constraints (Frantar et al., 2022; Yao et al., 2021; Liu et al., 2023b; Yuan et al., 2023; Zhang et al., 2025).

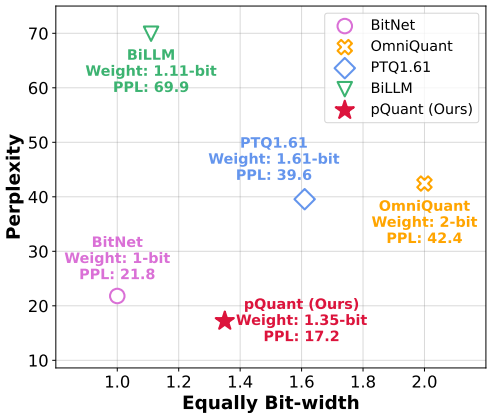


Figure 1: An overview of performance (Perplexity on WikiText2) and bit-width achieved by our pQuant and other extremely low-bit methods on 1.3B model.

Extremely low-bit (sub 2-bit) quantization stands out for offering the most aggressive model compression, promising drastic reductions in memory and compute footprint (Qin et al., 2022; Shang et al., 2023; Huang et al., 2024; Ma et al., 2024a; Xu et al., 2024). Its key hardware advantage lies in replacing costly floating-point matrix multiplications with efficient bitwise operations during inference, presenting a promising path toward edge-deployable LLMs (Zhu et al., 2024). Recent advances in extremely low-bit quantization have yielded notable empirical results. For instance, post-training quantization (PTQ) methods such as PTQ1.61 (Zhao et al., 2025) and BiLLM (Huang et al., 2024) achieve 68.5% and 61.2%, respectively, of FP16 baseline accuracy on downstream tasks. Quantization-aware fine-tuning further improves performance, with state-of-the-art approaches recovering up to 79.3% of FP16 accuracy (Xu et al., 2024).

Despite their strong efficiency benefits, extremely low-bit LLMs face limited practical adoption due to the accuracy degradation, less than 80% of FP16 capability. This challenge has spurred in-

terest in Quantization-Aware Training from scratch (QAT-Scratch), which trains low-bit models directly from random initialization without compressing pre-trained weights (Wang et al., 2023; Ma et al., 2024a, 2025). QAT-Scratch shows strong promise: BitNet achieves 90.1% FP16 parity on downstream tasks (Wang et al., 2023), and BitNet1.58 enables near-lossless 2-bit quantization with a 3B model (Ma et al., 2024b). Nevertheless, our experiments reveal two critical issues that persist in current 1-bit QAT-Scratch models. As Figure 1 shows: (i) even the state-of-the-art 1-bit method BitNet still exhibit a non-negligible accuracy gap; and (ii) scaling efficiency is poor: as model size increases, performance gains grow sublinearly and fall far behind those of FP16 models. This limits the practicality of extremely low-bit LMs in large-scale applications.

What underlies this performance and scaling bottleneck? It is well-established in quantization theory that parameters possess varying sensitivity, with a small subset of "sensitive" parameters disproportionately influencing model output (Dettmers et al., 2022, 2023; Lee et al., 2024; Li et al., 2025a). Surprisingly, we find that existing 1-bit QAT-Scratch models lose this structure: weight sensitivity becomes nearly uniform. We term this phenomenon parameter democratization, defined as the unintended homogenization of parameter sensitivity under extreme quantization. This observation motivates our central hypothesis: parameter democratization may be a key factor limiting the expressivity and scalability of 1-bit LMs.

To address and empirically validate our hypothesis on parameter democratization, we propose pQuant, a novel extremely low-bit QAT-Scratch method founded on the principle of decoupled and guided parameter sensitivity. The core of pQuant is a decoupled linear layer that structurally enforces parameter specialization: a 1-bit main branch ensures foundational efficiency, while a compact high-precision branch is dedicated to preserving sensitive parameters. Critically, importance assignment is not predetermined. Instead, feature scaling guide the model to dynamically allocate its most influential representations to the high-precision pathway. Build upon this, we further evolve the high-precision branch into a sparsely activated expert module, where a lightweight router selects one expert per token. This design enables substantial capacity growth with near-constant inference cost. Experiments show that pQuant reduces perplexity by 32.0% over state-of-the-art 1-bit baselines, and the high-precision branch effectively preserves the most sensitive parameters. When scaled, it surpasses 2-bit models in accuracy while improving inference throughput by 18.2%. Notably, it matches the performance of FP16 models while delivering over 2× higher throughput.

2 Preliminary

2.1 Quantization Aware Training

Quantization-Aware Training improves model robustness to quantization by simulating low-bit operations during training. Standard QAT fine-tunes a pre-trained full-precision model, using FP16 shadow weights to compute gradients while quantizing weights and activations in the forward pass (Shen et al., 2021). Recent advances enhance this paradigm: LLM-QAT (Liu et al., 2023b) employs data-free distillation; BitDistiller (Du et al., 2024) introduces asymmetric clipping; EfficientQAT (Chen et al., 2024) reduces cost via staged optimization; GETA (Qu et al., 2025) jointly optimizes pruning and quantization. Technical details of QAT are provided in Appendix J.2. While these methods achieve strong results at 4–8 bit, they face fundamental challenges when pushed to sub-2-bit regimes, often struggling to recover accuracy due to the difficulty of compressing pre-trained high-precision representations.

2.2 Extremely Low-Bit Quantization

Extremely low-bit (sub 2-bit) quantization aims for the most hardware-efficient LLMs but suffers from severe accuracy degradation. Early binarization methods like BNN (Hubara et al., 2016) and XNOR-Net (Rastegari et al., 2016) (which introduced scaling factors) do not scale to modern LLMs. Recent work has adapted quantization for LLMs: PB-LLM (Shang et al., 2023) identifies salient weights, BiLLM (Huang et al., 2024) proposes a two-stage binarization framework, OneBit (Xu et al., 2024) uses SVID decomposition, and BinaryMoS (Jo et al., 2024) employs token-adaptive scales. PTQ1.61 (Zhao et al., 2025) introduces a one-dimensional structured mask to reduce the upper bound of quantization errors,

A promising direction is Quantization-Aware Training from scratch (QAT-Scratch), which trains low-bit models from random initialization. BitNet (Wang et al., 2023) pioneered this paradigm for 1-bit LLMs, showing significantly better perfor-

mance than PTQ or fine-tuning-based QAT. BitNet1.58 (Ma et al., 2025) extended this success to 2-bit, achieving near-lossless accuracy. Other works like FBI-LLM (Ma et al., 2024a) (using progressive distillation) and iFairy (Wang et al., 2025) (using complex-valued representations) further explore this training paradigm. However, as identified in our work, existing QAT-Scratch methods still face a fundamental bottleneck in expressivity and scalability, which we attribute to the parameter democratization effect.

2.3 Sensitivity Analysis

Not all parameters in a neural network contribute equally to its output. Intuitively, a weight is considered sensitive to quantization if it incurs a large rounding error (i.e., it lies far from a quantization grid point) and/or if it frequently multiplies large input activations, thereby amplifying even small errors (Dettmers et al., 2023; Lee et al., 2024; Shang et al., 2023; Li et al., 2025b).

Inspired by SPQR (Dettmers et al., 2023), we adopt a perturbation-based metric to analyze such sensitivity under extreme low-bit settings. For a weight w_{ij} in matrix W and given calibration inputs X, we define its sensitivity as the minimum increase in squared output distortion:

$$s_{ij} = \min_{W'} \|WX - W'X\|_2^2, \tag{1}$$

where W' satisfies $w'_{ij} = \text{quant}(w_{ij})$, and all other entries of W' are free to adjust for optimal error compensation. Crucially, this problem admits a closed-form solution under the generalized Optimal Brain Surgeon framework (Frantar and Alistarh, 2022):

$$s_{ij} = \frac{w_{ij}^2}{2(XX^\top)^{-1}}, \tag{2}$$

where $(XX^\top)^{-1}$ is the inverse Hessian matrix corresponding to the optimization problem. This term quantifies how "replaceable" the weight is given the correlation structure of the inputs. Thus, sensitivity s_{ij} captures a joint effect: a large rounding error w_{ij}^2 may still yield low sensitivity if the weight lies in a direction where the input allows easy compensation. This saliency measure can be approximated efficiently by quantization solvers, such as GPTQ(Frantar et al., 2022).

To analyze the sensitivity landscape under extreme quantization, we set $\text{quant}(w_{ij}) = 0$ to perturb and compute per-layer sensitivity using a small calibration set X. As shown in Figure 2, this

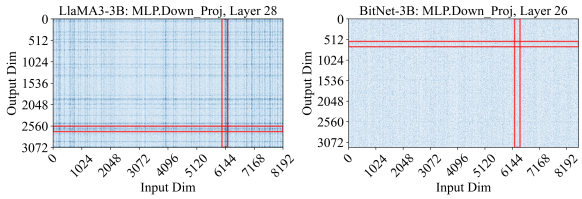


Figure 2: Weight log-sensitivities in the final FFN layer of LLaMA3-3B and BitNet-3B. Matrices are downsampled via max pooling for visualization; darker blue indicates higher sensitivity. Red boxes highlight regions of peak sensitivity. Notably, in the 1-bit weights of BitNet-3B, no pronounced sensitivity variation is observed.

reveals a stark contrast: in the 16-bit LLaMA-3 model, a small subset of parameters exhibits significantly higher sensitivity, indicating their disproportionately large influence on the output. In contrast, the 1-bit BitNet model shows a flattened, near-uniform distribution, suggesting that all parameters are treated with comparable importance.

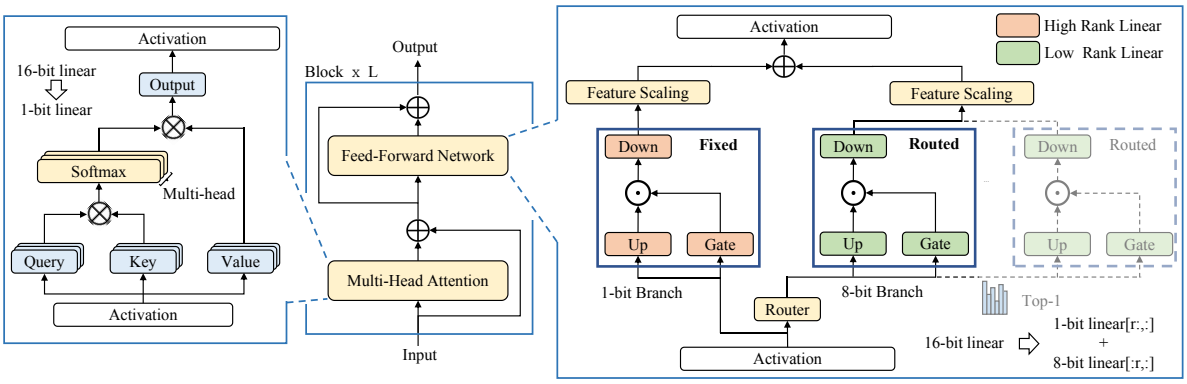
We refer to this observed collapse of sensitivity differentiation as parameter democratization. This homogenization may prevent the model from effectively prioritizing the most informative features. We therefore hypothesize that recovering a differentiated sensitivity structure, one in which critically important parameters are distinctly identified and preserved, could be key to improving both the accuracy and scalability of extremely low-bit LLMs.

3 Methodology

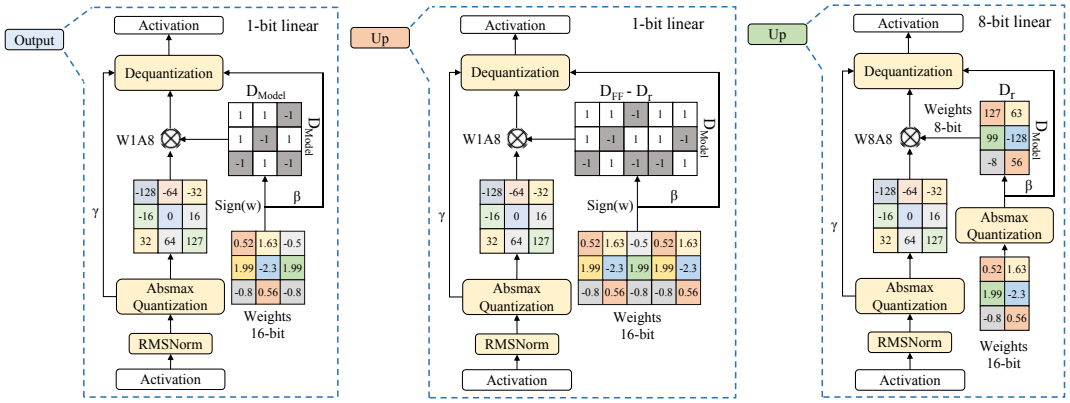
We propose pQuant, a method designed to counteract parameter democratization by structurally decoupling parameters, isolating a small set of sensitive weights into a higher-precision branch within an efficient 1-bit backbone. As illustrated in Figure 3, pQuant replaces standard linear layers in both multi-head attention (MHA) and feed-forward network (FFN) modules with quantization-native layers and uses feature scaling to explicitly guide importance-aware learning.

Within the MHA module, pQuant substitutes the query, key, value, and output projections with a pure 1-bit linear layer (Figure 3(b)). For the FFN, we design a decoupled linear layer: most parameters are quantized to 1-bit for efficiency (Figure 3(c)), while a small subset is kept in 8-bit to preserve highly sensitive weights (Figure 3(d)).

This decoupled design can be naturally viewed through the lens of a Mixture of Experts (MoE): the 1-bit branch acts as a shared expert common to all inputs, ensuring efficiency, while the 8-bit



(a) The architecture of the decoupled design in pQuant



(b) 1-bit Liner in MHA (c) 8-bit Linear in FFN (d) 1-bit Linear in FFN

Figure 3: Computational flow of pQuant's core modules. (a) pQuant replaces all linear layers with quantized counterparts, with 8-bit branch to enable dynamic scaling when needed. (b-d) Computation in three representative pQuant modules. FP16 weights are retained solely during training to ensure numerical stability and discarded post-training. Here, r denotes the dimension of weights in 8-bit branch, where $r \ll D_{\text{model}}$.

branch functions as a specialized, routable expert. A lightweight, fixed top-1 router dynamically activates routable expert based on the input token.

3.1 Multi-Head Attention in pQuant

In the MHA, each linear layer utilizes 1-bit weights represented with INT1. To maximize inference efficiency, we apply this aggressive, undifferentiated quantization specifically to MHA, while reserving our decoupled architecture for the FFN. This targeted design is supported by two key observations from prior work. First, FFN layers are known to concentrate a larger fraction of sensitive parameters that are critical to model performance (Dettmers et al., 2023). Second, activation distributions in FFNs tend to be less regular and exhibit more outliers compared to the relatively structured activations in attention layers (Shazeer, 2020).

During pretraining, weights and activations are dynamically quantized to low precision, while gradients and optimizer states are maintained in FP32 to preserve training stability and accuracy. Specifically, FP16 weights are used in the training pass to ensure meaningful gradient updates. Using 1-bit weights during optimization would result in near-zero gradients for most parameters, making it difficult for the model to learn effective parameter updates and reduce the loss. These weights are discarded during inference, leaving only the 1-bit parameters in the model.

These FP16 weights are quantized to 1-bit before performing the tensor multiplication between weights and activations. After the multiplication, a dequantization process is applied to the output. This process can be formally described as follows:

$$W^{INT1} = Sign(W^{Float} - \mu) \tag{3}$$

$$Sign(W_{ij}) = \begin{cases} +1, & \text{if } W_{ij} > 0 \\ -1, & \text{if } W_{ij} < 0 \end{cases} \tag{4}$$

$$Y = \lambda \cdot W^{INT1} LayerNorm(X) \tag{5}$$

$$\mu = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}, \lambda = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} |W_{ij}| \tag{6}$$

Here, $W^{INT1} \in \mathbb{R}^{m \times n}$ denotes the 1-bit quantized weight matrix, and W^{float} the FP16 reference. X and Y are the input and output of the linear layer, respectively. To reduce the ℓ_2 quantization error, a dequantization scale λ is applied. Following Liu et al. (2022), we center the weights to zero mean using μ prior to binarization, which enhances the information capacity of binary weights. Additionally, pQuant quantizes all activations to INT8 using AbsMax scaling, as detailed in Appendix A.

3.2 Feed-Forward Network in pQuant

To preserve highly sensitive parameters with higher precision, a straightforward approach is to designate a subset of dimensions in the linear layer to be represented in high precision while keeping the rest in 1-bit. However, this naïve strategy faces two inherent limitations in the QAT-Scratch setting. First, since parameters are randomly initialized and trained from scratch, there exists no consistent spatial pattern in the weight matrix that reliably indicates where sensitive parameters reside (Yu et al., 2024). Predefining fixed positions for high-precision weights thus diminishes the potential benefit. Second, the non-uniform distribution of input activation magnitudes can dilute the advantage of retaining higher precision in arbitrary dimensions, often rendering the improvement marginal—similar to random selection. These issues highlight the need for a more adaptive and learnable strategy for high-precision allocation.

Accordingly, we introduce the decoupled linear layer, which structurally splits the original weight matrix in the FFN into two parallel computational branches: a 1-bit branch and a high-precision branch. This design allows the high-precision branch to operate independently, effectively avoiding the dynamic range collapse issue inherent in per-tensor 1-bit quantization and AbsMean scaling. Both branches process the same input activations and their outputs are then summed to form the final result.

In pQuant, we adopt 8-bit precision for the high-precision branch, motivated by extensive evidence that 8-bit representations preserve essential parameter information while maintaining computational efficiency (Liu et al., 2024a; Peng et al., 2023; Van Baalen et al., 2023; Mishra et al., 2025). Among 8-bit formats, we specifically choose INT8 over FP8 or MXFP8 due to its superior hardware support and deployment compatibility on mainstream CPU/GPU platforms—practical advantages critical in extreme low-bit regimes.

A naive summation of the two branches' outputs would conflate their distinct sensitivity profiles, hindering the model's ability to learn their relative importance. Inspired by RSLoRA and MS-LoRA (Kalajdzievski, 2023; Luo et al., 2025), which demonstrate that scaling informative components enhances representational capacity allocation. We introduce *feature scaling*, where the outputs of the two FFN branches are modulated by learnable scalars α and β. To prioritize the high-precision signal during optimization, we initialize $\alpha \gg \beta$ (e.g., $\alpha = 2.0$, $\beta = 0.2$), ensuring the 8-bit branch receives stronger gradient feedback during backpropagation. The FFN computation is then:

$$Y = \alpha\, FFN^{INT8}_{[:r]}(LayerNorm(X)) + \beta\, FFN^{INT1}_{[r:]}(LayerNorm(X)), \quad (7)$$

where r is a hyperparameter controlling the fraction of high-precision weights retained to preserve sensitivity-critical parameters, with $r \in [0, D_{\text{ffn}}]$ and D_{ffn} denoting the FFN hidden dimension. The 1-bit weights follow the same quantization scheme as in the MHA's 1-bit linear layers (§3.1), while the 8-bit weights are quantized identically to 8-bit activations (Appendix A).

3.3 Efficient Scaling of pQuant

We expands the high-precision branches into multiple parallel branches and employs a lightweight router to dynamically activate the most suitable path based on the input features. This structure serves as an optional mechanism for performance enhancement. Specifically, pQuant extend the number of 8-bit branch to N, incorporating a gating to select the most appropriate weights for input activations, as shown in Figure 3(a). The resulting FFN architecture bears a resemblance to Mixture-of-Experts (e.g., DeepSeekMoE (Dai et al., 2024) and OpenMoE (Xue et al., 2024)). However, the two designs are fundamentally different due to the discrepancy in precision and dimension.

The router in pQuant is implemented as a simple linear layer and trained end-to-end along with the rest of the model. pQuant adopt the top-1 strategy for selecting the 8-bit branch, ensuring that the total number of parameters active in the FFN matches that of a conventional FFN. The gating function employs a softmax activation (Shazeer et al., 2017). The N acts as a hyperparameter, with detailed analysis presented in (§4.3).

Parameter	D_{Model}	D_{FF}	r	1-bit	8-bit
300M	1024	2272(2400-128)	128	96%	4%
700M	1536	3840(4096-256)	256	96%	4%
1.3B	2048	5076(5400-384)	384	95%	5%
2.6B	2880	7168(7680-512)	512	95%	5%

Table 1: Configurations for model trained by pQuant. $\mathbf{D_{FF}}$ denote the hidden dimension of the feed-forward network; r denotes the dimension of the 8-bit branch; 1-bit and 8-bit denote the percentage of total parameters represented by 1-bit and 8-bit respectively.

4 Experiments

In this section, we conduct extensive experiments to validate our method pQuant. We further evaluate the inference efficiency. Finally, we show the ablation studies of pQuant.

pQuant is trained end-to-end via QAT-Scratch, adopting a two-stage schedule to ensure stable and robust convergence (Ma et al., 2024b). Gradients for quantized weights during training are approximated using the Straight-Through Estimator (STE) (Bengio et al., 2013) during backpropagation. Training details are provided in Appendix C. We also optimize the inference efficiency of pQuant, as detailed in Appendix B.

4.1 Experimental setup

In our experimental, we train pQuant at four scales detailed in Table 1. For a comprehensive comparison, we also train from scratch three representative baselines under identical configurations: BitNet (1-bit model) (Wang et al., 2023), BitNet1.58 (2-bit model) (Ma et al., 2024b), and LLaMA-3 (16-bit model) (Touvron et al., 2023b). Additionally, we compare pQuant against post-training quantization (PTQ) methods, including PTQ1.61 (Zhao et al., 2025) and OmniQuant (Shao et al., 2023), as well as fine-tuning-based QAT approaches such as OneBit (Xu et al., 2024). Our models are trained on three datasets: C4 (Raffel et al., 2020), Wikipedia (Foundation, 2020) and ArXiv (Cohan et al., 2018). We configured the batch size to 4 million tokens for LlaMA-2 and 1 million tokens for the other models.

We evaluate the models based on their zero-shot performance in some downstream tasks, including ARC-Easy (ARC-E) (Yadav et al., 2019), ARC-Challenge (ARC-C) (Yadav et al., 2019), BoolQ(BQ) (Clark et al., 2019), PIQA(PQ) (Bisk et al., 2020), Winogrande(WGe) (Sakaguchi et al., 2021), OpenbookQA(OQ) (Mihaylov et al., 2018), and Hellaswag(HS) (Zellers et al., 2019). More details about experimental setup is shown in Appendix D.

4.2 Performance of pQuant

The main evaluation results are summarized in Table 2. Compared to the BitNet (1-bit), pQuant (1.35-bit) achieves consistent gains across all metrics. Notably, the 700M-parameter pQuant outperforms the larger 1.3B BitNet on both downstream accuracy and perplexity, suggesting that preserving sensitive weights in a high-precision branch improves parameter efficiency.

At the matched 1.3B scale, pQuant narrows the gap to the BitNet1.58 (2-bit) to just 0.4 average accuracy points, despite using 0.65 fewer bits per weight. pQuant surpasses two strong baselines of the same scale: OmniQuant (2-bit) and OneBit (1-bit), both derived from OPT-1.3B pretrained on 180B tokens (80% more than our datasets).

Most strikingly, when scaled to 2.6B parameters, pQuant achieves an average accuracy of 47.1, surpassing the 1.3B LLaMA-2 (FP16) on all evaluated tasks—and substantially outperforming PTQ1.61, the current SOTA extremely low-bit PTQ method, even when applied to the much larger LLaMA-2-7B. The lower PPL is attributed to PTQ1.61's model being trained on significantly more data, far exceeding our 100B tokens. This demonstrates that pQuant not only maintains performance under extreme quantization but also scales effectively.

4.3 Scalability of pQuant

We scale pQuant by increasing the number of high-precision expert branches N from 1 to 8, while maintaining the same number of active parameters per forward pass as a standard FFN. As shown in Figure 4, at $N = 8$, pQuant consistently outperforms BitNet1.58 in training loss across all model sizes. More importantly, the performance gap between pQuant and the FP16 LLaMA-2 baseline narrows as model size increases: at 1.3B parameters, pQuant nearly matches LLaMA-2's loss, whereas 1-bit BitNet and 2-bit BitNet1.58 exhibit a persistent gap. This contrast highlights a key finding: scaling via pure 1-bit parameters is inherently inefficient under extreme quantization. In contrast, pQuant's strategy of scaling high-precision branches enables it to follow a scaling law much closer to that of full-precision models, further evidencing its effective scalability. Downstream accuracy and perplexity results are reported in Ta-

Parameters	Method	Bit	ARC-E↑	ARC-C↑	HS↑	BQ↑	OQ↑	PQ↑	WGe↑	Avg.↑	Perplexity↓
300M	FP16	16	44.8	19.3	34.8	60.7	18.6	66.8	51.3	42.3	22.9
	BitNet	1	38.2	16.2	32.9	53.5	14.2	58.1	48.8	37.6	34.2
	pQuant	1.28	41.5	19.7	33.0	60.0	16.2	62.5	50.9	40.5	30.1
700M	FP16	16	51.4	21.4	37.3	58.9	19.0	68.5	52.4	44.1	16.5
	BitNet	1	44.4	18.7	35.4	56.5	14.9	62.9	50.0	40.4	27.6
	pQuant	1.28	47.0	20.1	36.2	57.2	17.4	67.1	51.6	42.4	21.9
1.3B	FP16	16	53.9	22.1	38.7	55.7	21.8	71.0	54.3	45.4	14.4
	BitNet	1	47.8	19.2	36.3	58.6	17.6	67.6	52.0	42.6	21.8
	BitNet1.58	2	50.4	21.2	36.9	61.2	19.4	68.7	53.0	44.4	16.9
	OmniQuant	2	38.8	23.4	33.4	56.5	15.3	60.9	51.9	40.0	42.43
	OneBit	1.1	41.3	24.1	34.3	59.5	16.3	62.6	51.1	41.3	25.4
	pQuant	1.35	49.8	20.8	**37.0**	60.2	**20.1**	68.3	52.5	44.0	17.2
2.6B	**pQuant**	1.35	54.6	22.9	40.5	61.6	22.0	71.9	56.4	47.1	13.0
7B	PTQ1.61	1.61	47.2	22.3	35.8	56.5	15.3	63.2	52.3	41.8	12.7

Table 2: Main experimental results. We report perplexity and zero-shot accuracy of BitNet (1-bit), BitNet1.58 (2-bit), FP16 LLaMA-2 (16-bit), and pQuant on downstream tasks. Perplexity is evaluated on WikiText-2. Results show that pQuant significantly improves the performance of 1-bit models.

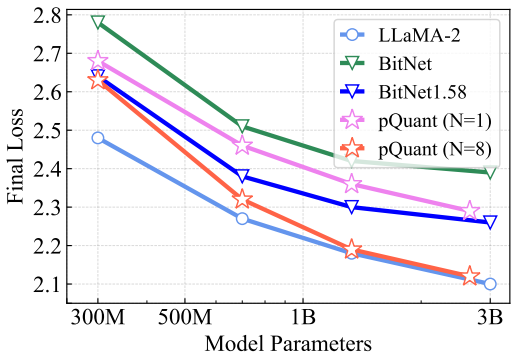


Figure 4: Final training loss with varying numbers of parameters. pQuant with N=8 demonstrates excellent scalability, whereas 1-bit BitNet do not.

ble 5. The overhead introduced by adding more 8-bit branches remains minimal; a detailed analysis is provided in Appendix E.

4.4 Sensitivity Distribution of pQuant

To validate that pQuant effectively mitigates parameter democratization, we analyze the weight sensitivity distribution in the final FFN layer of pQuant-1.3B after training, as shown in Figure 5(a). Compared to the uniform sensitivity distribution observed in BitNet-3B, pQuant exhibits a markedly differentiated sensitivity profile, with distinct regions of high and low sensitivity. This contrast confirms that pQuant's decoupled architecture and feature scaling successfully guide the model to allocate sensitivity-critical parameters effectively, thereby enhancing expressivity and performance.

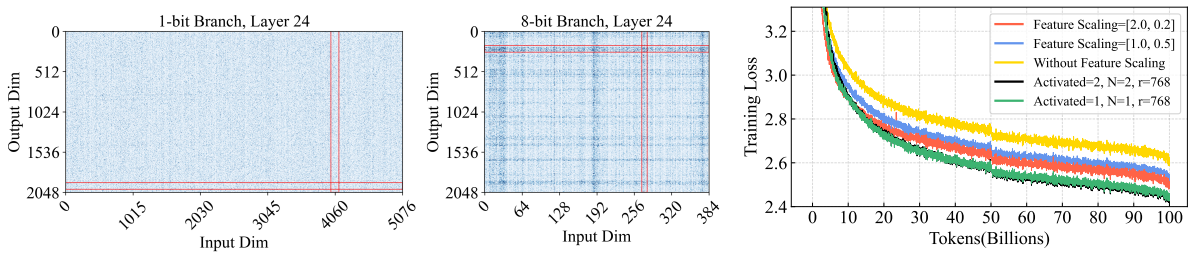
4.5 Inference Efficiency of pQuant

We evaluate the end-to-end inference throughput of LLaMA-2, BitNet1.58, and pQuant on an Apple M2 platform with a 7B model. pQuant attains a throughput of 133.1 tokens/s, which is 2.0× that of LLaMA-2 (65.3 tokens/s) and outperforms BitNet1.58 (112.6 tokens/s) by 18.2%. Computation and memory efficiency of pQuant are detailed in Appendix B.

4.6 Ablation Study

Feature Scaling. As show in Figure 5 (b), we initialize the learnable factory as $\alpha = 1.0$ and $\beta = 0.5$. Post-training analysis reveals distinct converged values (mean $\alpha \approx 2.0$, $\beta \approx 0.2$); re-initializing to these values improves performance. Crucially, models trained under different scaling configurations do *not* converge to similar final losses, indicating that feature scaling exerts a persistent structural influence, rather than merely shaping early optimization dynamics. Ablating feature scaling leads to a significant increase in training loss, confirming its necessity for effective allocation of sensitivity-critical parameters. Further analysis is provided in Appendix G.

High-precision Branch. For hardware efficiency, we restrict the 8-bit branch dimension r to integer multiples of 128. First, under a single-branch setting, increasing r from 256 to 768 (at 700M scale)



(a) Sensitivity analysis of pQuant

(b) Ablation on pQuant

Figure 5: (a) Sensitivity analysis of the 1-bit and 8-bit branches in the down-projection layer of the final FFN block in the 700M pQuant model. (b) Ablation study of pQuant on 700M-parameter model, assessing the impact of feature scaling and the number of active 8-bit branches on final loss. The mid-training loss drop stems from the two-phase learning rate schedule (see AppendixC.2).

Model	Total	Activated	PPL ↓	Memory
pQuant($N = 4$)	1.5B	1.3B	15.5	0.91G
BitNet1.58	1.3B	1.3B	16.9	0.72G
pQuant($N = 8$)	1.3B	926M	16.8	0.98G
LlaMA-2	1.3B	1.3B	14.4	2.64G

Table 3: Performance comparison under Matched Parameter. Memory footprint include the storage of Embeddings and LayerNorm.

improves performance, but the gains do not compensate for the added memory and computation overhead. Second, we evaluate a multi-branch strategy that activates two branches of size $r = 768$ per token, which yields negligible improvement.

In our experiments, pQuant with $N = 4$ achieves performance better than BitNet1.58. To rigorously evaluate the architectural efficiency of pQuant, we conduct a controlled comparison under matched parameter budgets. Specifically, we reduce the hidden state dimension of pQuant to match the total of 1.3B parameters in BitNet1.58. We configure pQuant with $N = 8$ and 926M activated parameters, using only one active branch. Under these settings, pQuant runs 1.6× faster than BitNet1.58 duirng inference. The results are shown in Table 3. When evaluated on the same test set as our main experiments, both models achieve comparable performance. This parity in results, achieved despite pQuant's reduced hidden dimension, provides compelling evidence for pQuant's superior parameter efficiency and architectural advantages in low-bit quantization scenarios.

Additional ablation studies, such as the effect of varying the 8-bit branch size, ablating the 8-bit branch entirely, and learning rate, are provided in Appendix F.

4.7 Limitations

Although QAT-Scratch methods, including pQuant, gain significant advantages in model performance, they generally incur higher training costs compared to traditional QAT and PTQ. The training duration across different models is presented in Appendix I.

While pQuant can achieve better inference speed and model performance than 2-bits BitNet1.58 when $N \geq 4$, this configuration introduces higher physical memory requirements, as shown in subsection E.1. The speed improvement stems primarily from reduced memory access volume. Our design aligns with current trends in edge-device hardware development, where memory capacity is more cost-effective than bandwidth. This memory overhead represents a meaningful trade-off that merits further investigation.

Due to the training cost, our experiments are limited to models size. Future work could explore scaling pQuant to larger models (e.g., 70B) to further validate its effectiveness.

5 Conclusion

We propose pQuant, a quantization-aware training-from-scratch method for extremely low-bit LLMs. Motivated by the observation that existing 1-bit models suffer from *parameter democratization*, i.e., the homogenization of weight sensitivity that limits expressivity and scaling, we design a decoupled linear layer that preserves sensitive weights in a small high-precision branch. Coupled with feature scaling to prioritize gradient flow toward informative components, this approach enables effective allocation of model capacity under extreme quantization. Experiments show that pQuant not only surpasses prior extremely low-bit methods at matched scales but also scales effectively.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2021. Understanding and overcoming the challenges of efficient transformer quantization.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2024. Quip: 2-bit quantization of large language models with guarantees.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. 2024. Efficientqat: Efficient quantization-aware training for large language models.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, and 1 others. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression.

Dayou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. 2024. Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation.

Wikimedia Foundation. 2020. Wikimedia downloads.

Elias Frantar and Dan Alistarh. 2022. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023. A framework for few-shot language model evaluation.

Wei Huang, Yangdong Liu, Haotong Qin, Ying Li, Shiming Zhang, Xianglong Liu, Michele Magno, and Xiaojuan Qi. 2024. Billm: Pushing the limit of post-training quantization for llms.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Advances in neural information processing systems*, 29.

Dongwon Jo, Taesu Kim, Yulhwa Kim, and 1 others. 2024. Mixture of scales: Memory-efficient token-adaptive binarization for large language models. *Advances in Neural Information Processing Systems*, 37:137474–137494.

Damjan Kalajdzievski. 2023. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:13355–13364.

Guoyu Li, Shengyu Ye, Chunyun Chen, Yang Wang, Fan Yang, Ting Cao, Cheng Liu, Mohamed M Sabry Aly, and Mao Yang. 2025a. Lut-dla: Lookup table as efficient extreme low-bit deep learning accelerator.

He Li, Jianhang Hong, Yuanzhuo Wu, Snehal Adbol, and Zonglin Li. 2024. Continuous approximations for improving quantization aware training of llms.

Shiyao Li, Yingchun Hu, Xuefei Ning, Xihui Liu, Ke Hong, Xiaotao Jia, Xiuhong Li, Yaqi Yan, Pei Ran, Guohao Dai, and 1 others. 2025b. Mbq: Modality-balanced quantization for large vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 4167–4177.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report.

Shih-Yang Liu, Zechun Liu, and Kwang-Ting Cheng. 2023a. Oscillation-free quantization for low-bit vision transformers.

Zechun Liu, Barlas Oguz, Aasish Pappu, Lin Xiao, Scott Yih, Meng Li, Raghuraman Krishnamoorthi, and Yashar Mehdad. 2022. Bit: Robustly binarized multi-distilled transformer. *Advances in neural information processing systems*, 35:14303–14316.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023b. Llm-qat: Data-free quantization aware training for large language models.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024b. Spinquant: Llm quantization with learned rotations.

Dan Luo, Kangfeng Zheng, Chunhua Wu, and Xiujuan Wang. 2025. Mslora: Meta-learned scaling for adaptive fine-tuning of lora. *Neurocomputing*, page 130374.

Ziyang Luo, Artur Kulmizev, and Xiaoxi Mao. 2020. Positional artefacts propagate through masked language model embeddings.

Liqun Ma, Mingjie Sun, and Zhiqiang Shen. 2024a. Fbi-llm: Scaling up fully binarized llms from scratch via autoregressive distillation.

Shuming Ma, Hongyu Wang, Shaohan Huang, Xingxing Zhang, Ying Hu, Ting Song, Yan Xia, and Furu Wei. 2025. Bitnet b1. 58 2b4t technical report.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. 2024b. The era of 1-bit llms: All large language models are in 1.58 bits.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering.

Asit Mishra, Dusan Stosic, and Simon Layton. 2025. Recipes for pre-training llms with mxfp8.

Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2024. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models.

Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, and 1 others. 2023. Fp8-lm: Training fp8 large language models.

Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua Yan, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. 2022. Bibert: Accurate fully binarized bert.

Xiaoyi Qu, David Aponte, Colby Banbury, Daniel P Robinson, Tianyu Ding, Kazuhito Koishida, Ilya Zharkov, and Tianyi Chen. 2025. Automatic joint structured pruning and quantization for efficient neural network training and compression. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 15234–15244.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Yuzhang Shang, Zhihang Yuan, Qiang Wu, and Zhen Dong. 2023. Pb-llm: Partially binarized large language models.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models.

Noam Shazeer. 2020. Glu variants improve transformer.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.

Mingzhu Shen, Feng Liang, Ruihao Gong, Yuhang Li, Chuming Li, Chen Lin, Fengwei Yu, Junjie Yan, and Wanli Ouyang. 2021. Once quantization-aware training: High performance extremely low-bit architecture search.

Ma Shuming, Wang Hongyu, and Furu Wei. 2024. The era of 1 bit llms training tips code faq.

Falcon-LLM Team. 2025. Falcon-e, a series of powerful, universal and fine-tunable 1.58bit language models.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Mart Van Baalen, Andrey Kuzmin, Suparna S Nair, Yuwei Ren, Eric Mahurin, Chirag Patel, Sundar Subramanian, Sanghyuk Lee, Markus Nagel, Joseph Soriaga, and 1 others. 2023. Fp8 versus int8 for efficient deep learning inference.

Feiyu Wang, Guoan Wang, Yihao Zhang, Shengfan Wang, Weitao Li, Bokai Huang, Shimao Chen, Zihan Jiang, Rui Xu, and Tong Yang. 2025. ifairy: The first 2-bit complex llm with all parameters in {+1, -1, +i, -i}. *arXiv preprint arXiv:2508.05571*.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. 2023. Bitnet: Scaling 1-bit transformers for large language models.

Jinheng Wang, Hansong Zhou, Ting Song, Shaoguang Mao, Shuming Ma, Hongyu Wang, Yan Xia, and Furu Wei. 2024a. 1-bit ai infra: Part 1.1, fast and lossless bitnet b1. 58 inference on cpus.

Lei Wang, Lingxiao Ma, Shijie Cao, Quanlu Zhang, Jilong Xue, Yining Shi, Ningxin Zheng, Ziming Miao, Fan Yang, Ting Cao, Yuqing Yang, and Mao Yang. 2024b. Ladder: Enabling efficient Low-Precision deep learning computing through hardware-aware tensor transformation. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 307–323, Santa Clara, CA. USENIX Association.

Jianyu Wei, Shijie Cao, Ting Cao, Lingxiao Ma, Lei Wang, Yanyong Zhang, and Mao Yang. 2024. T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge. *Preprint*, arXiv:2407.00088.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. 2023. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. 2022. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems*, 35:17402–17414.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models.

Yuzhuang Xu, Xu Han, Zonghan Yang, Shuo Wang, Qingfu Zhu, Zhiyuan Liu, Weidong Liu, and Wanxiang Che. 2024. Onebit: Towards extremely low-bit large language models.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. 2024. Openmoe: An early effort on open mixture-of-experts language models.

Vikas Yadav, Steven Bethard, and Mihai Surdeanu. 2019. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, and 1 others. 2021. Hawq-v3: Dyadic neural network quantization.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. 2019. Understanding straight-through estimator in training activation quantized neural nets.

Mengxia Yu, De Wang, Qi Shan, and Alvin Wan. 2024. The super weight in large language models.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, and 1 others. 2024. Llm inference unveiled: Survey and roofline model insights.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence?

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. 2025. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30.

Jiaqi Zhao, Miao Zhang, Ming Wang, Yuzhang Shang, Kaihao Zhang, Weili Guan, Yaowei Wang, and Min Zhang. 2025. Ptq1. 61: Push the real limit of extremely low-bit post-training quantization methods for large language models. *arXiv preprint arXiv:2502.13179*.

Rui-Jie Zhu, Yu Zhang, Ethan Sifferman, Tyler Sheaves, Yiqiao Wang, Dustin Richmond, Peng Zhou, and Jason K Eshraghian. 2024. Scalable matmul-free language modeling.

A Activation Quantization

pQuant employ the AbsMax method (Dettmers et al., 2024) to quantization all activations and high-precision branch in FFN along the token dimension to the range $[-2^7, 2^7]$, and represent them in INT8 format:

$$Q(X) = RoundClip(X \times \gamma, -2^7 + \epsilon, 2^7 + \epsilon) \quad (8)$$

$$RoundClip(X, a, b) = \max(a, \min(b, Round(X)), \quad (9)$$

$$\gamma = \frac{2^7}{\max(|x_1|, |x_2|, \ldots, |x_n|)} \quad (10)$$

where $Round(\cdot)$ function rounds each value to the nearest integer. The parameter ϵ is a small floating-point value that prevents overflow during clipping. Additionally, γ is the scaling factory for activations, which will be incorporated with λ to de-quantize the output as follows:

$$Y = \frac{\lambda}{\gamma} \times W^{INT1} Q(LayerNorm(X)) \quad (11)$$

B Inference

Although pQuant introduces additional operations such as quantizaiton, de-quantizaiton, rescale and layernorm during training that slightly increase training overhead, these costs are entirely eliminated during inference, leading to highly efficient deployment. First, the parameters in 1-bit branch are offline quantized and stored in 1-bit precision during inference, matching the effect of PTQ methods. They are packed into UINT8 format with 8 parameter per byte, resulting in a storage footprint that is 1/16 of that required by FP16 models. This also reduces memory bandwidth when GPU cores load weight matrices from global memory. Second, the scaling factors used during training can be fused into preceding or succeeding layers during inference. For example, per-tensor scaling factors can be integrated into the input normalization of the subsequent module. Similarly, the RMSNorm operation can be merged with activation quantization, as both are element-wise transformations. These optimizations eliminate the need for half-precision in GEMM (General Matrix Multiplications) operations, allowing us to employ mixed-precision operators that support W1A8 GEMM computations within linear (Wang et al., 2024b). Third, in the first GEMM of the FFN layer during inference, the same input must be multiplied with both the 8-bit and 1-bit branches of the up projection. To improve efficiency, this computation can be distributed across multiple thread groups, enabling parallel execution without redundant data reads.

In pQuant, the acceleration method is built atop T-MAC (Wei et al., 2024), a CPU-based inference library that natively supports mixed-precision matrix multiplication without dequantization. 1-bit quantization is particularly valuable in edge-device deployment, where the batch size is typically one and the most time-consuming operation becomes GEMV (General Matrix-Vector Multiplication) rather than GEMM. A promising approach to implement bitwise computation is through table lookups (Park et al., 2024). Since each 1-bit weight can only take two values, the number of possible bit patterns in a small group is limited. For instance, if a 1-bit matrix is partitioned into groups of four elements, there are only 2^4 possible combinations per group (e.g., [1,1,1,-1], [1,1,-1,-1]). Given an activation vector, the results of its multiplication with all possible bit patterns can be precomputed and stored in a lookup table. The mixed-precision GEMM between activation and 1-bit weights is then transformed into a series of table lookups indexed by the bit patterns in the weight, followed by summation of the retrieved results. This reduces GEMM to table lookup and addition operations, eliminating multiplications entirely.

We evaluate the end-to-end inference throughput of LLaMA-2, BitNet1.58, and pQuant on an Apple M2 platform with a 7B model. pQuant attains a throughput of 133.1 tokens/s, which is 2.0× that of LLaMA-2 (65.3 tokens/s) and outperforms BitNet1.58 (112.6 tokens/s) by 18.2%. Although both compute and memory bandwidth improvements exceed 5.6×, the end-to-end speedup is only slightly above 2.0×, primarily due to model size: in workloads dominated by frequent vector and small matrix multiplications, scheduling overhead constitutes a significant fraction of execution time, whereas larger models can achieve 5.1× higher throughput (Wang et al., 2024a).

B.1 Computation Efficiency

The use of 1-bit weights eliminates a significant amount of matrix multiplication operations, thereby substantially reducing computational complexity (Zhu et al., 2024). To further evaluate its practical efficiency, we conducted comprehen-

sive benchmarking of overhead during inference. Figure 6 presents the time costs for linear operation across different models. Compared to BitNet1.58 and LlaMA-2, pQuant demonstrates a 82% improvement in computational performance over LlaMA-2 and is notably more efficient than BitNet1.58, reducing computation time by 38%.

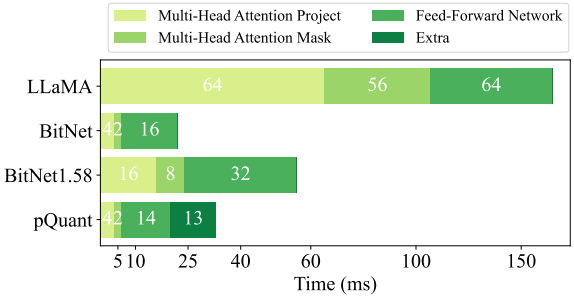


Figure 6: Computation time across components in a Transformer block, measured on an Apple M2 with a 7B model and input sequence length of 256. pQuant is 38% faster than BitNet1.58 and 82% faster than FP16 LLaMA-2.

B.2 Memory Efficiency

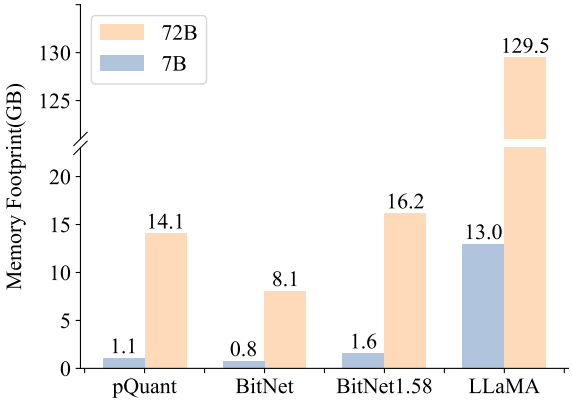


Figure 7: The memory footprint. Memory footprint of model weights transferred during a single forward pass in inference. pQuant is smaller than BitNet1.58 and significantly lower than LLaMA-2.

In addition to computation, the process of transferring model parameters from DRAM to the memory of an on-chip accelerator (e.g., SRAM) can be expensive during inference (Yuan et al., 2024). Compared to half-precision models, 1-bit models also suggests the opportunity to improve the serving latency. Thus, we outline the required memory for various model sizes in Figure 7. pQuant maintains a consistent memory footprint during decoding regardless of the value of N, as it activates only a single 8-bit branch at any given time. This design avoids excessive memory accesses and better leverages the characteristics of modern hardware, where bandwidth is often the primary bottleneck rather than memory capacity. Compared to LlaMA-2, pQuant reduces memory usage by 92%, and requires 31% less memory than BitNet1.58. In pQuant, numerous individual scalars are introduced: $alpha$, $beta$, $gamma$, mu, and $lambda$. These scalars can be merged during inference, ensuring that memory efficiency is not impacted.

C Training Mechanism

We introduce RMSNorm to compress the dynamic range of activations. This helps maintain basic performance under absmean-based quantization and has been shown to accelerate convergence in 1-bit quantized models (Ma et al., 2024b; Team, 2025). This provides a smoother optimization landscape for gradient descent and improves convergence (Liu et al., 2023a). The data is preprocessed using the BPE tokenizer with a vocabulary size of 32K. We select 500 steps for warm-up.

C.1 Straight-Through Estimator

Since the function sign(·) is non-differentiable at zero, it causes the gradient chain to break during backpropagation, preventing optimization of the model parameters. Therefore, we use the Straight-Through Estimator (STE) method during back propagation. In traditional backpropagation, gradients flow through differentiable functions allowing for the optimization of weights via gradient descent. However, when operations are not continuous or involve discrete decisions, calculating these gradients becomes problematic. The STE provides a way around this by passing the gradient straight through the operation during the backward pass while using the quantized values in the forward pass (Li et al., 2024). In essence, STE approximates the gradient of a non-differentiable function as 1, allowing the network to learn despite the presence of non-differentiable operations.

C.2 Two-Phase Training Schedule

Besides, pQuant employ a two-phase training schedule (Shuming et al., 2024). In the first phase, the learning rate begins at a high value and decreases linearly until the midpoint of training. The second phase starts with a lower learning rate,

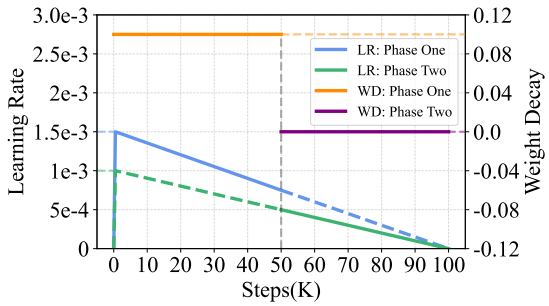


Figure 8: Two-Stage Scheduler. During mid-training, both the learning rate (LR) and weight decay (WD) are decayed to accelerate convergence of 1-bit parameters and prevent sign flips caused by excessive weight oscillation around quantization thresholds.

which also decays linearly. Besides, weight decay is set to 0.1 during the first phase and disable in the second phase. These strategies have been shown to improve convergence in low bit training, as demonstrated by BitNet1.58. This approach results in a significant reduction in the loss curve shortly after the midpoint. The learning rates and weight decay for the two phases are shown in the Figure 8.

There are two primary motivations for this design. First, in contrast to FP16 models where learning rates are typically well calibrated, 1-bit optimization faces a key challenge: small updates to the latent weights often fail to produce meaningful changes in the quantized 1-bit weights. This results in biased gradient estimation and suboptimal update behavior, particularly during the early training stages when rapid convergence is critical. Second, in half-precision training, weight decay serves as an effective regularizer by constraining large weight magnitudes, thereby preventing overfitting and improving training stability. However, in 1-bit training, weight decay is applied to the latent weights rather than the actual 1-bit weights, which alters its regularization effect and may lead to unintended consequences.

D Experiment Details

These common sense reasoning benchmarks we choosed in our experiments have been widely adopted in prior work to assess quantization effectiveness, in both post-training quantization (PTQ) and quantization-aware training (QAT) methods (Xiao et al., 2023; Lin et al., 2024; Chee et al., 2024; Shang et al., 2023; Ashkboos et al., 2024). We also use perplexity to quantitatively measure the model's generation power on WikiText-2 (Merity et al., 2016). We followed the pipeline from lm-evaluation-harness4 (Gao et al., 2023) to perform the evaluation.

The configurations for the BitNet, BitNet1.58, and LlaMA-2 models are shown in Table 4. The sequence length was uniformly set to 2048. To optimize training efficiency and performance, we employed mixed-precision training, and the Adam optimizer is applied with $\beta_1 = 0.9$ and $\beta_2 = 0.95$.

Our models were trained using 16 NVIDIA A100-80G GPUs and 1TB of CPU memory. We utilized the DeepSpeed framework (Rasley et al., 2020) for training.

Parameters	D_{Model}	D_{FF}	Heads	Layers
300M	1024	2400	16	24
700M	1536	4096	24	24
1.3B	2048	5460	32	24

Table 4: Configurations for BitNet, BitNet1.58 and LlaMA-2. Head denotes the number of attention heads in multi-head attention; Layers denotes the number of Transformer blocks.

E Analysis of Scaling pQuant

The main results in Table 2 suggests that merely increasing the parameter count is insufficient for extracting additional information in the extremely low-bit scenario. The expressive capacity of 1-bit tensor is highly constrained under the absmean quantization scheme, limiting its scalability.

The Table 5 below presents the complete performance test results after scaling.

E.1 Overhead of Model Parameter

Although the number of activated parameters during inference in pQuant is consistent with that of other models, the overall parameter count increases with varying N values. Specifically, the total parameter counts for models of different sizes, when N is 1, 2, 4 or 8 are shown in the Table 6. When N=1, pQuant has the same number of parameters as normal transformers model.

F Ablation Study

High-Precision Branch. We also evaluate the performance of pQuant without the 8-bit branch, training the model from scratch using QAT. As shown in Table 7, we compare the perplexity of a 300M pQuant with 8-bit branch ($N = 1$) and without it ($N = 0$), where the latter setting resembles

Models	Total/Activated	ARC-E↑	ARC-C↑	HS↑	BQ↑	OQ↑	PQ↑	WGe↑	Avg.↑	Perplexity↓
LlaMA-2 (16-bit)	300M	**44.8**	19.3	**34.8**	**60.7**	**18.6**	**66.8**	51.3	**42.3**	**22.9**
BitNet1.58 (2-bit)	300M	43.8	19.7	34.6	57.5	16.0	64.9	52.2	41.2	29.7
pQuant(N=8)	366M/300M	43.5	**20.1**	34.5	60.2	17.2	64.6	**51.9**	41.7	27.1
LlaMA-2 (16-bit)	700M	**51.4**	**21.4**	**37.3**	**58.9**	**19.0**	68.5	52.4	**44.1**	**16.5**
BitNet1.58 (2-bit)	700M	48.5	20.0	35.9	57.7	17.8	68.3	51.1	42.8	21.1
pQuant(N=8)	898M/700M	48.5	20.7	36.7	58.6	18.6	**68.5**	**52.6**	43.5	17.9
LlaMA-2 (16-bit)	1.3B	**53.9**	**22.1**	38.7	55.7	**21.8**	**71.0**	54.3	45.4	**14.4**
BitNet1.58 (2-bit)	1.3B	50.4	21.2	36.9	**61.2**	19.4	68.7	53.0	44.4	16.9
pQuant(N=8)	1.7B/1.3B	52.9	21.7	**38.9**	60.0	21.6	70.3	**55.5**	**45.8**	14.3

Table 5: Average score on downstream tasks. Results show that pQuant achieves efficient scaling by increasing the capacity of the high-precision branch. When the number of 8-bit branch reaches 8, pQuant surpasses 2-bit baselines and matches the accuracy of FP16 models. Total/Activated means total parameters/activated parameters per forward pass.

Base Size	N=1	N=2	N=4	N=8
300M	300M	309M	328M	366M
700M	700M	728M	785M	898M
1.3B	1.3B	1.4B	1.5B	1.7B

Table 6: Total parameters of pQuant. N denotes the number of 8-bit branches used.

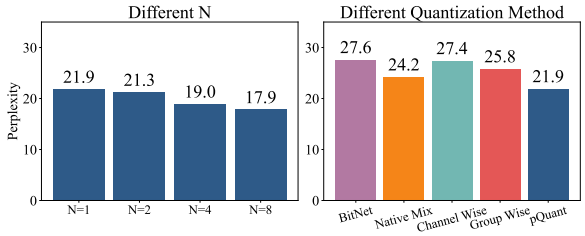


Figure 9: Perplexity of pQuant under different configurations. Lower perplexity indicates better performance. Left: Perplexity decreases as the number of 8-bit branches N in pQuant increases. Right: Comparison with alternative quantization methods. Native Mix retains 8% of parameters in high precision on top of 1-bit BitNet, rather than using our decoupled architecture. Channel-wise quantizes weights per output channel of the matrix multiplication. Group-wise quantizes weights in groups of 64 elements.

a dimension-reduced BitNet. The results show a significant performance gap. Combined with the analysis from OneBit (Xu et al., 2024), this supports our earlier claim: the impact of increasing rank in fully 1-bit models is limited, suggesting that we are approaching the performance limit of pure 1-bit models under such setting. Additionally, we evaluated the practical performance using FP16 for 8-bit parameters and found that the loss curve closely matches that of the setting with one activated 8-bit branch and $r = 576$, indicating that 8-bit precision, combined with a scaling factor, is already sufficient to represent the sensitive parameters in our architecture.

Model	300M w/	300M w/o	700M w/	700M w/o
Perplexity↓	30.1	36.9	21.9	29.3

Table 7: Performance comparison w/ or w/o 8-bit branch. Removing the 8-bit branch and training with only the 1-bit branch leads to a significant drop in model performance.

We show the number of 8-bit branches N ranging from 1 to 8, at 700M in the left of Figure 9. As N increases, theperformance of pQuant improves steadily. Specifically, in our tests, pQuant outperformed 2-bit model when $N = 4$. Moreover, as shown in the right panel of Figure 9, we explored directly keeping 8% of parameters in high precision from the start of BitNet training. This approach yields significantly worse performance than pQuant, despite using more high-precision parameters (8% vs. pQuant's 5%). We also evaluated channel-wise and group-wise quantization. Channel-wise quantization provides negligible gains. Group-wise quantization (with block size 64) achieves better accuracy but requires storing one 16-bit scaling factor per 64 1-bit weights, which incurs substantial metadata overhead and is unfriendly to hardware. These results highlight the effectiveness of pQuant.

Batch Size. We also evaluated various batch sizes and found 1 million tokens led to significant performance improvements compared to 4 million to-

kens. This indicates that low-bit models benefit more from frequent parameter updates to achieve optimal performance, rather than relying on larger batch sizes for stability.

Learning rate. As illustrated in Figure 5(b), we observed a significant reduction in loss occurring when the learning rate was decayed, rather than at the end of the training phase. This approach made the performance more predictable during the training process. However, half-precision models did not benefit from a similar learning rate decay strategy, likely because their loss curves do not exhibit an S-shaped pattern.

G Feature Scaling Analysis

As illustrated in Table 8, the feature scaling values of 1.3B pQuant at the end of training exhibit two clear patterns: (1) both the initial and final layers exhibit relatively large scaling magnitudes, while the intermediate layers show smaller values. (2) the scaling factors for the 8-bit branches are significantly higher than those for the 1-bit branches, which aligns with our design principle of allocating higher precision to more influential weights.

H Stability of pQuant

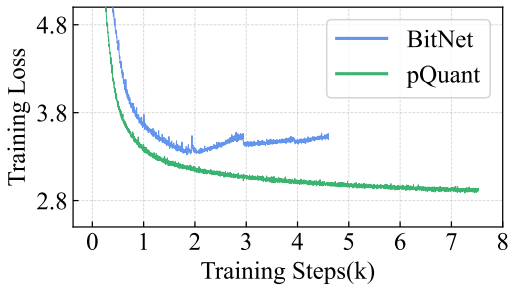


Figure 10: Training loss curve of two model. pQuant exhibits greater training stability. In contrast, BitNet frequently suffers from gradient explosion during training, often requiring checkpoint reloading and restarts.

Additionally, we observed that the 1-bit model experienced instability and crashes at a batch size of 4 million tokens, which is the batch size used for LlaMA-2. Even under the optimal batch size of 1M suggested in the BitNet research, training often becomes unstable. As shown in the Figure 10, this necessitates frequent rollbacks to previous checkpoints to resume training. In contrast, pQuant did not show similar issues under the same conditions, indicating its superior stability.

I Training Time

The gradient update process in QAT models differs from that in traditional pretraining, often involving a simpler or more constrained optimization landscape. pQuant follows this characteristic. This is primarily due to the additional computational steps involved in quantization, dequantization, and auxiliary matrix operations, which are currently implemented using high-precision arithmetic. The consequence of latent weight keeping during trainin is that we are keeping both a FP16 and 1-bit copy of weights during training. This makes training less memory efficient than in a standard Transformer. Since large models are bottlenecked on memory bandwidth, this has a significant impact on training performance.

The estimated training duration across different models is presented in Table 9.

J Related Work

J.1 Post-Training Quantization

Current quantization approaches are broadly categorized into Quantization-Aware Training and Post-Training Quantization. QAT incorporates quantization and dequantization during the training phase, enabling the model to adapt to the constraints imposed by reduced precision. In contrast, PTQ applies quantization without requiring further training.

A number of recent studies (Luo et al., 2020; Bondarenko et al., 2021; Wei et al., 2023; Xiao et al., 2023) have identified the existence of significant outliers in large language models. Outlier Suppression (OS) (Wei et al., 2022) restructures the LayerNorm function by shifting its scale parameter γ to reduce outlier effects. OS+ (Wei et al., 2023) introduces a channel-wise shifting transformation that balances activation magnitudes across channels to handle asymmetric distributions. SmoothQuant (Xiao et al., 2023) applies a per-channel scaling transformation that redistributes quantization difficulty from activations to weights. AWQ (Yao et al., 2021) observes that the importance of weight channels is largely determined by activation scales. The idea of rotation transformation was first explored in QuIP (Chee et al., 2024), later extended by QuaRot (Ashkboos et al., 2024), which integrates quantization for both weights and activations, including the KV cache. GPTQ (Frantar et al., 2022) adopts a more accurate quantization framework and achieves strong results in 4-bit settings.

Layers	1	2	3	4	6	8	10	12	14	16	18	20	21	22	23	24
8-bit	1.82	1.73	1.57	1.59	1.39	1.33	1.25	1.22	1.34	1.49	1.37	1.27	1.38	1.45	1.56	1.61
1-bit	0.78	0.62	0.48	0.42	0.38	0.34	0.31	0.26	0.29	0.36	0.43	0.45	0.41	0.47	0.52	0.50

Table 8: Feature scaling after training. In the trained model, feature scaling reveals the model's preference for the 8-bit branch, highlighting its ability to preserve outliers.

Parameters	N=1	N=2	N=4	N=8
300M	1.9	2.0	2.1	2.3
700M	4.9	5.1	5.4	6.0
1.3B	8.5	8.8	9.6	11.1

Table 9: Total days for training.

SpinQuant (Liu et al., 2024b) finds that different rotation matrices can lead to varying performance in quantized models and proposes a learning-based approach to optimize rotation transformations. ZeroQuant (Yao et al., 2022) and LLM.int8() (Dettmers et al., 2022) improve quantization accuracy by introducing custom grouping strategies for quantization blocks. SpQR (Dettmers et al., 2023) further partitions features to better manage precision constraints. OWQ (Lee et al., 2024) preserves information capacity in sensitive weight channels through carefully designed scaling transformations.

J.2 Quantization-Aware Training

In QAT, the training process typically involves three key components: quantization simulation, gradient approximation, and parameter optimization. During forward propagation, weights are passed through a quantize-dequantize pipeline to mimic the effect of low bit width computation in real deployment. Specifically, they are first quantized into lower precision representations and then dequantized back to higher precision values for further computation. Since quantization operations such as rounding are non differentiable (Yin et al., 2019), gradient approximation techniques are employed during backpropagation. With this setup, the model learns to optimize its parameters under simulated quantization conditions, thereby improving robustness to quantization noise (Shen et al., 2021). As illustrated in Figure 11, given a 16-bit weight matrix, a quantization scale is typically derived from its dynamic range (e.g., maximum absolute value). The matrix is then scaled, rounded to 1-bit integers, and dequantized back to floating point. The difference between the dequantized and original matrices constitutes the quantization error.

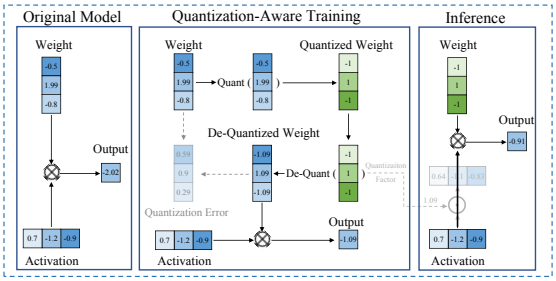


Figure 11: Illustration of the Quantization-Aware Training process. Before quantization, matrix multiplications are performed in 16-bit precision. During QAT, simulated quantization are inserted to enable the model to learn and compensate for quantization-induced errors. In the final inference phase, quantization factors are fused into the weights, only use the quantized weights.

The effectiveness of QAT largely depends on the design of the pseudo-quantization operation, which varies in how well it reduces model sensitivity to quantization. A well-designed pseudo-quantization mechanism allows the model to adjust its weights during training in response to quantization effects, making it more robust or "numb" to the distortions introduced during actual quantization.